Отчёт по лабораторной работе №9

Архитектура компьютера

Иванов Сергей Владимирович

Содержание

# 1 Цель работы

Целью лабораторной работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Выполнение лабораторной работы

Создадим каталог для программ лабораторной работы № 9, перейдем в него и создадим файл lab9-1.asm.(рис. [1](#fig:001)).

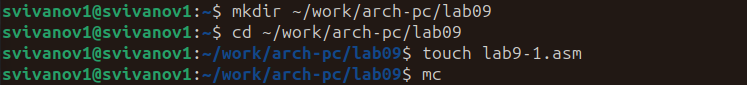


Figure 1: Файл lab9-1.asm

Введем в файл программу листинга 9.1, создадим исполняемый файл и проверим его работу.(рис. [2](#fig:002)).

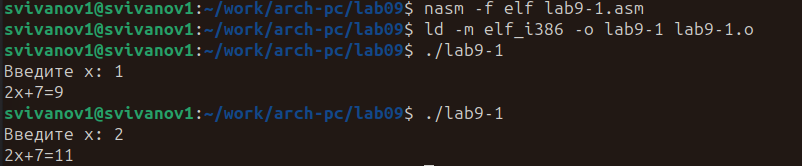


Figure 2: Работа файла lab9-1.asm

Изменим текст программы добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. (рис. [3](#fig:003)).

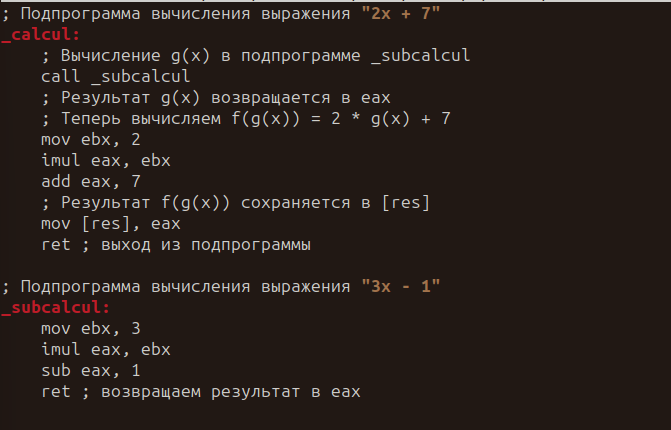


Figure 3: Изменение файла

Создадим исполняемый файл и запустим его, программа работает корректно. (рис. [4](#fig:004)).

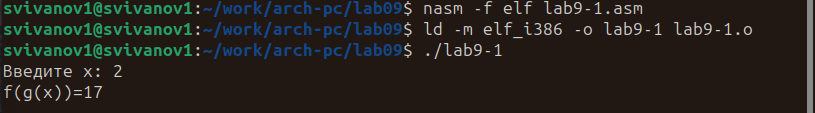


Figure 4: Результат работы файла lab9-1.asm

Создадим файл lab9-2.asm с текстом программы из Листинга 9.2. Получим исполняемый файл. Загрузим исполняемый файл в отладчик gdb. Проверим работу программы, запустив ее в оболочке GDB с помощью команды run(рис. [5](#fig:005)).

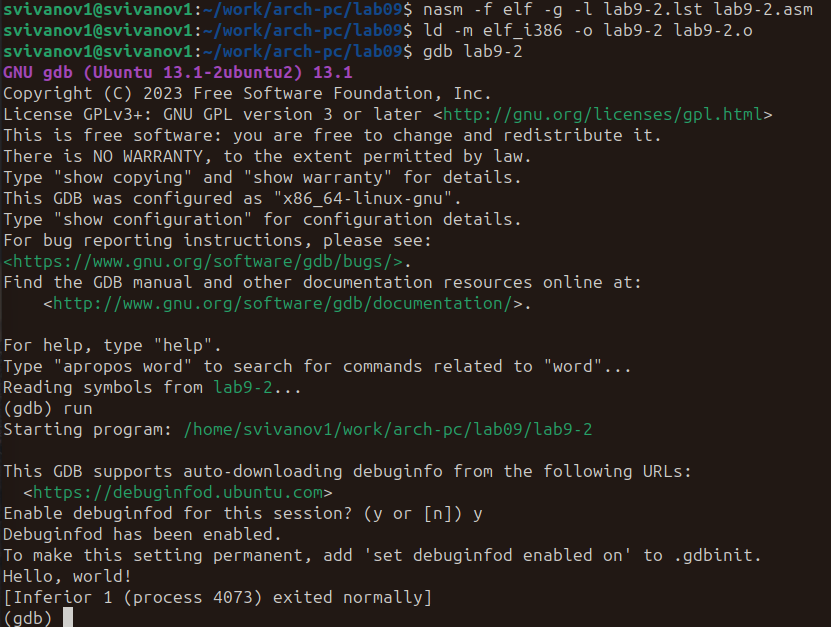


Figure 5: Запуск lab9-2.asm в gdb

Установим брейкпоинт на метку \_start, и запустим её(рис. [6](#fig:006)).

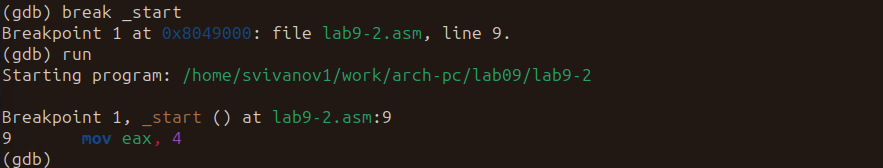


Figure 6: Брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel(рис. [7](#fig:007)).



Figure 7: Дисассимилированный код

Pазличия отображения синтаксиса машинных команд в режимах ATT и Intel: противоположное расположение операнда-источника и операнда-приемника; в ATT регистры пишутся после ‘%’, а непосредственные операнды после ‘$’, в синтаксисе Intel операнды никак не помечаются.

Включим режим псевдографики для более удобного анализа программы(рис. [8](#fig:008)).

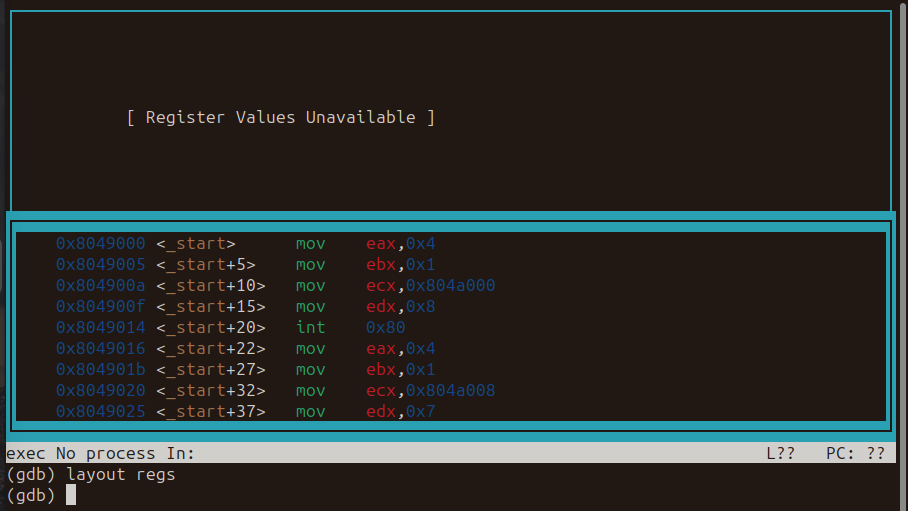


Figure 8: Pежим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды info breakpoints. Установим еще одну точку останова по адресу предпоследней инструкции. Посмотрим информацию о всех установленных точках останова(рис. [9](#fig:009)) .

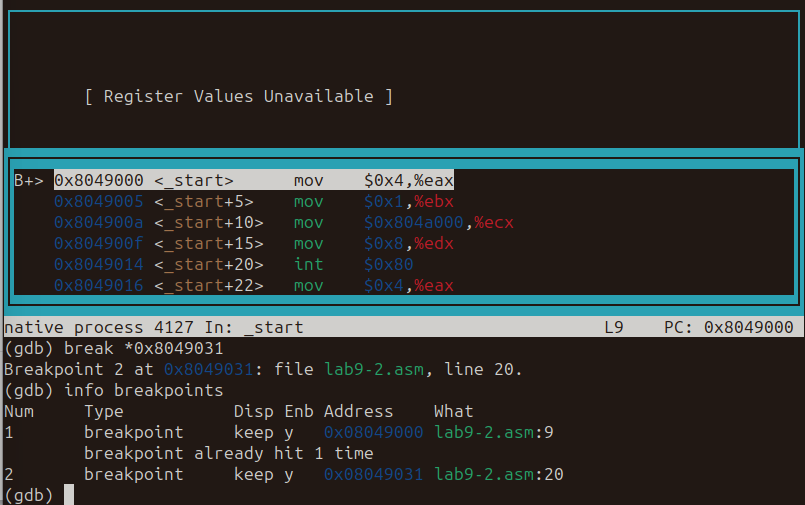


Figure 9: info breakpoints

Выполним 5 инструкций с помощью команды stepi и проследим за изменением значений регистров. В результате изменяются значения регистров eax, ebx, ecx, edx(рис. [10](#fig:010)).

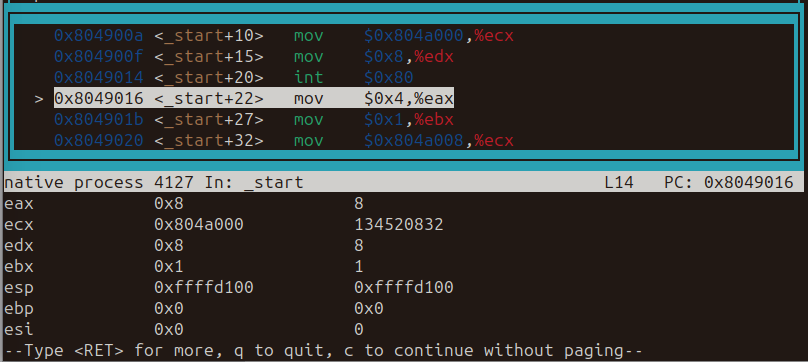


Figure 10: Команда stepi

Посмотрите значение переменной msg1 и msg2 по имени. Изменим первый символ переменной msg1(рис. [11](#fig:011)).

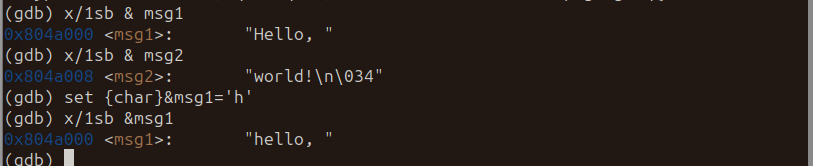


Figure 11: Изменение символа

Заменим символ во второй переменной msg2.(рис. [12](#fig:012)).

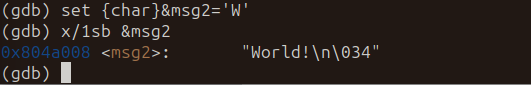


Figure 12: Изменение символа в msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [13](#fig:013)).

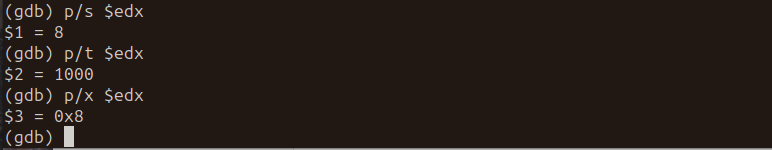


Figure 13: Регистр edx

С помощью команды set изменим значение регистра ebx: (рис. [14](#fig:014)).



Figure 14: Значения регистра ebx

Использовав команду set изменили значение регистра ebx сначала на символ ‘2’, а затем на число 2, и сравнили вывод значения регистра в десятичном формате. В результате присвоения регистра значение символа ‘2’, выводится число 50, что соответствует символу в ‘2’ в таблице ASCII

Cкопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab9-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы: (рис. [15](#fig:015)).

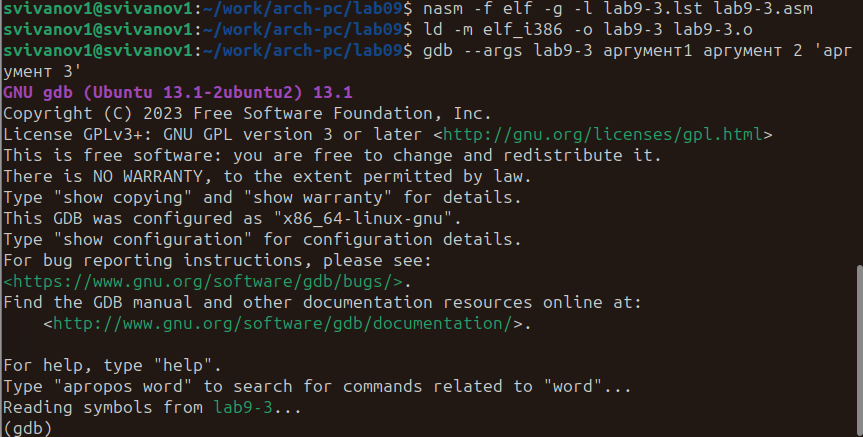


Figure 15: Файл lab9-3.asm

Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. [16](#fig:016)).

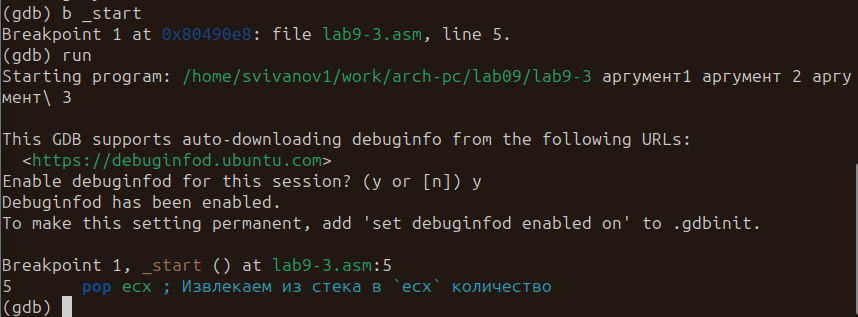


Figure 16: Адрес вершины стека

Посмотрим остальные позиции стека. (рис. [17](#fig:017)).

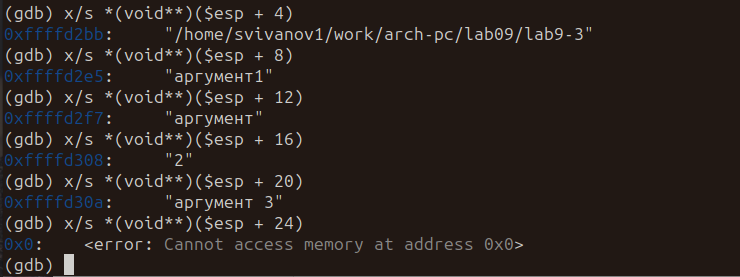


Figure 17: Все вершины стека

В первом хранится адрес, в остальных хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт:каждый элемент стека занимает 4 байта, поэтому для получения следующего элемента стека мы добавляем 4 к адресу вершины.

# 3 Задание для самостоятельной работы.

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. Запустим файл. Программа работает корректно. (рис. [18](#fig:018)).

%include 'in\_out.asm'  
  
SECTION .data  
 msg db "Результат: ", 0  
 msg1 db "Функция: f(x)=7+2x", 0  
  
SECTION .text  
global \_start  
  
; Подпрограмма для вычисления функции f(x)  
calculate\_f:  
 ; Вход: eax - значение x  
 ; Выход: eax - результат функции f(x)  
 add eax, eax ; умножение x на 2  
 add eax, 7 ; добавление 7  
 ret  
  
\_start:  
 mov eax, msg1  
 call sprintLF  
 pop ecx ; Извлекаем из стека в ecx количество аргументов  
 pop edx ; Извлекаем из стека в edx имя программы  
  
 sub ecx, 1 ; Уменьшаем ecx на 1 (количество аргументов без названия программы)  
 mov esi, 0 ; Используем esi для хранения промежуточных сумм  
  
next:  
 cmp ecx, 0h ; Проверяем, есть ли еще аргументы  
 jz \_end ; Если аргументов нет, выходим из цикла  
  
 pop eax ; Иначе извлекаем следующий аргумент из стека  
 call atoi ; Преобразуем символ в число  
 call calculate\_f ; Вызываем подпрограмму для вычисления f(x)  
 add esi, eax ; Добавляем к промежуточной сумме следующий аргумент (esi = esi + f(x))  
 loop next ; Переход к обработке следующего аргумента  
  
\_end:  
 mov eax, msg ; Вывод сообщения "Результат: "  
 call sprint  
 mov eax, esi ; Записываем сумму в регистр eax  
 call iprintLF ; Печать результата  
 call quit ; Завершение программы

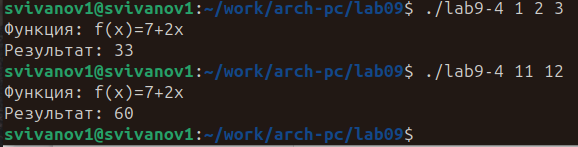


Figure 18: Программа lab9-4

1. Создадим файл с текстом из листинга 9.3, где приведена программа вычисления выражения (3+2)\*4+5. При запуске данная программа дает неверный результат

Проанализируем с помощью отладичка GDB изменение значений регистров, чтобы найти ошибку. Установим брейкпоинт b \_start, включим режим псевдографики и начнём поочердно выполнять команды и следить за значениями регистров. Ошибка в том, что результат 3+2=5 записан в регистре ebx (команда add ebx, eax), но команда mul всегда перемножает значение регистра eax с указанным сомножителем, поэтому mul ecx дает неверный результат. Затем, к нему добавляется число 5, результат запоминается в регистре edi, а затем выводится (рис. [19](#fig:019)).

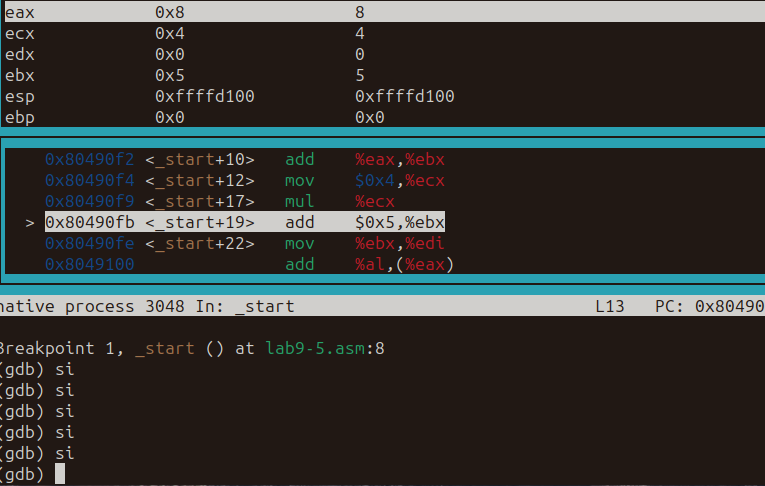


Figure 19: Поиск ошибок с помощью отладчика

Исправим текст программы:

%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov ebx,3  
mov eax,2  
add eax,ebx  
mov ecx,4  
mul ecx  
add eax,5  
mov edi,eax  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

Запустим файл. Программа работает корректно.(рис. [20](#fig:020)).

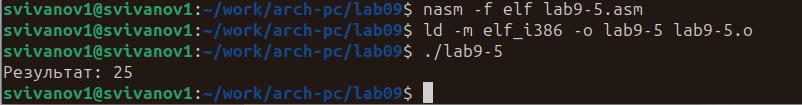


Figure 20: Запуск файла

# 4 Выводы

В результате выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.